UCRL- 84658
PREPRINT

SIGNAL PROCESSING ASPECTS OF THE S-1 MULTIPROCESSOR PROJECT

John L. Manferdelli
P. Michael Farmwald
William Bryson

This paper was prepared for submittal to:
SPIE Annual International Technical Symposium,
San Diego, Society of Photo Optical
Instrumentation Engineers, July 30, 1980

July 28, 1980

Signal Processing Aspects Of The S-1 Multiprocessor Project

P. Michael Farmwald, William Bryson and John L. Manferdelli

University of California, Lawrence Livermore National Laboratory
Livermore, CA 94550

Abstract

The S-1 family of uni- and multiprocessors is an extremely high performance but relatively inexpensive general purpose set of digital processing systems being developed for the most demanding National security applications. A typical S-1 system consists of from one to sixteen S-1 uniprocessors sharing up to 4 gigawords of uniformly addressed main memory. Each uniprocessor of the current (Mark IIA) generation has computational power roughly equivalent to that of a CRAY-1. In this paper, we give a general description of the S-1 Mark IIA uniprocessor and a somewhat more extensive discussion of its signal processing hardware and hardware-executed algorithms. Our focus is a discussion of the felicitous properties of the uniprocessor components that make the S-1 system a powerful general purpose digital signal processor.

S-1 System Architecture

A basic goal of the S-1 Mark IIA design has been to produce a flexible supercomputer with special signal processing capabilities. The strategy employed in realising this goal has been to design a high performance uniprocessor to be used as either as a stand-alone CPU or as a member of a closely coupled network consisting of multiple processors interconnected into a memory of enormous (by previous standards) capacity. Unlike many current signal processors, an S-1 uniprocessor incorporates its general purpose functions in the same unit that houses the special purpose high speed arithmetic/signal processing functions; this allows uniform access to signal processing functions without the performance degradation associated with ancillary processors (e.g., that due to slow bus speeds and transmission of data from general purpose main memory to special purpose "boxes").

Space limitations (and the reader's patience) do not permit a complete description of the designs of S-1 uniprocessor and multiprocessor systems here; instead, the interested reader may repair to [1] for a good overview, or to [2] for a complete description of the S-1 system design.

S-1 Instruction Set

The basic S-1 instruction set architecture was designed for high performance implementation, yet is oriented towards high-level language usage. It includes a vector and signal processing instruction subset, as well as a few special purpose instructions (e.g. ones for sorting). This combination of powerful elementary machine capabilities has several implications. It simplifies the efficient use of the the signal processing constructs from a high level language, since the complexity of generating code for two dissimilar machines running with two (or more) separate address spaces and differing instruction and data formats is avoided. In addition, the tight coupling between scalar and vector processing implies a greatly reduced "hardware dominance" effect on the choice of algorithms. An algorithm can be coded in a more natural or intuitive fashion, rather than having to be recast to fit the peculiarities of the architecture or implementation. As an example, the use of branches depending upon the values of just-computed data must be reduced to a minimum when using a signal processor, since it involves interaction between the host and the signal processor.

The large (two gigabyte) virtual address space of the S-1 architecture greatly simplifies its programming. Large data bases can thereby be conveniently maintained, and many applications can productively trade off memory capacity against real-time constraints. The cost of memory is dropping rapidly; we envision having main memory systems of up to 100 megabytes capacity by the end of 1981 (at a cost of no more than $5000 per megabyte).

Data Types

The S-1 architecture data formats include 9, 18, 36, and 72 bit signed and unsigned integers and 18, 36 and 72 bit floating-point numbers. These basic data types can be organised into more complicated types including complex numbers (pairs), vectors and arrays.

The three floating-point formats are halfword (one sign bit, five exponent bits and twelve fraction bits), singleword (one sign bit, nine exponent bits and 27 fraction bits), and doubleword (one sign bit, 15 exponent bits and 56 fraction bits). The S-1 floating-point data format also includes a "hidden" normalised bit. Since all S-1 floating-point numbers are normalised, it is redundant to include the most significant bit of the fraction. Thus the actual useful fractions are one bit larger than indicated.

In addition, the floating-point instructions treat certain values specially; this allows the maintainance of as much information as possible in the presence of overflow, underflow and division by zero. (The S-1 data formats are described in [2]; see [3] for a detailed description of a similar floating-point format.)

The 18-bit floating-point format is a rather novel data type for a general purpose computer; however, it promises to be the most useful data type for signal processing. In particular, it simplifies the computation of FFTs, since scaling is done automatically, and it also makes the calculation of low-precision FFTs more accurate [4] than can be obtained with an integer format of the same total data word size.

S-1 Mark IIA Uniprocessor Implementation

The time-consuming portions of most signal processing algorithms are simple parallel computations; however, as new machines become ever more parallel, the inherently serial or less structured computations will dominate execution time. The ability of a high performance signal processing architecture to do fast scalar processing will thus become as important as its ability to rapidly perform FFTs or vector multiplies. In addition, the plot of instruction rate vs. operation size must not have too large of a gap (Figure 1) between scalar (characterised by small size and little or no structure) and vector (characterised by arbitrary size and highly structured) operations; such a large gap in processing capability would make parallel but non-structured algorithms suffer greatly in performance, as has actually been seen in some recent superprocessor implementations.

The S-1 Mark IIA has attempted to address this evolving situation by having a balance of high performance scalar and vector processing capabilities. The Mark IIA uses the same hardware to execute both kinds of instruction, which leads rather naturally to a smooth trade-off between the instruction rate and the operation size.

The scalar instruction cycle time of the Mark IIA is 50 ns; the Mark IIA can therefore execute up to 20 million instructions per second. An S-1 scalar instruction can be quite powerful in that a single instruction may include indexing (including shifting of the indices), multiple memory or register reads (including the virtual-to-physical address translation) and finally execution of the operation itself, all pipelined at the rate of 50 ns per instruction. This is particularly important due to the high-level language orientation of most current systems development: many high level language constructs compile into single S-1 instructions. Of course, more complicated instructions don't pipeline at the maximum rate; some examples of these are subroutine calls or most types of divide instructions.

The operation of a pipeline is not always "smooth"; occasionally there are dependencies between nearby (in time) instructions which cause interlocks. The scalar latencies of the various instruction interlocks are complicated, due to the complex nature of the Mark IIA pipe. However, for most signal processing applications, the most interesting time is the latency of the "execution" of operations (i.e. the time from starting an addition operation to when the sum is available for use in the next operation). For most scalar operations of the Mark IIA, this quantity is readily determined. If the instruction is some form of multiplication (all precisions of multiplication and single precision reciprocation or square root), the latency is 150 ns. For most other "add class" operations, the execution latency is 100 ns. These "add

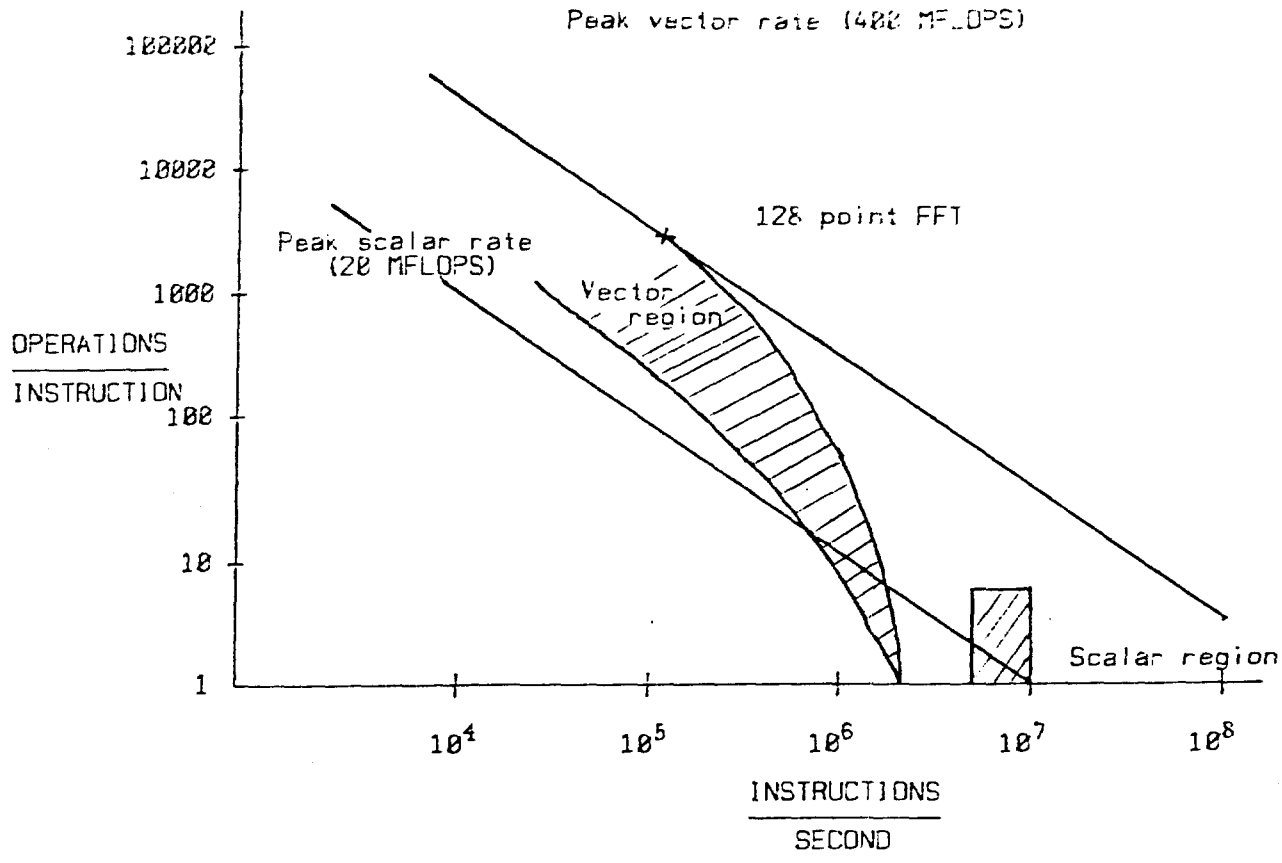

Figure 1. Graph of Operation rate vs. operation size.

class" operations include floating-point and integer addition and subtraction of all precisions, shifts, byte loads and deposits, minimum, maximum, absolute value, bit counting and "find first one bit". Other scalar operations take longer depending upon their complexity. Again it should be emphasized that these instructions all pipeline at 50 ns, the latency period is seen by the user only when there are nearby instruction dependencies.

A happy consequence of having such short scalar latencies is the ease with which they often can be eliminated by a small amount of code movement; thus there if there is at least one instruction between the "add class" instruction and the instruction which uses the "add class" result, the latency of the "add class" instruction is invisible and it will pipeline at 50 ns. Similarly, there need be only two instructions between the initiation of a multiplication and its use for the multiplication latency to be invisible.

The vector throughput of the Mark IIA can be as much 20 times greater than the peak scalar level (the vector halfword complex floating-point FFT does a complete butterfly every 25 ns, whereas the best scalar program could perform the same butterfly in 500 ns, i.e. a 1024-point complex FFT takes 0.13 ms, compared to 2.6 ms for a scalar equivalent); a better rule of thumb is that vector equivalents are four times faster than well-coded scalar loops. To give an better idea of the speeds involved, the loop

for $I := 1$ **to** $SIZE$ **do** $A[I] := B[I] + A[I]*D$;

is implemented by a single instruction and will require $[12.5\ SIZE + 250]$ ns to cmplete for halfwords (18-bits), $[25\ SIZE + 250]$ ns for singlewords (36-bits), and $[50\ SIZE + 250]$ ns for doublewords (72-bits).

The vector and special instructions are decoded using the same hardware as is used for scalar instructions; thus the rate of vector instructions is determined by the size of the operation plus a small additional overhead (0-300 ns, depending upon the instruction). The small size of the overhead is important; it implies that the performance is not degraded significantly when processing short vectors. In fact, for many vector operations, the execution time for a vector of length two is the same as for the equivalent scalar code.

Table 1 lists some representative instructions and their execution times; tables 2 and 3 lists some vector and signal processing times.

Instruction (H-halfword, S-singleword, D-doubleword)	Pipeline Time	Latency
Move (HSD)	50 ns	100 ns
Shift (HSD)	50 ns	100 ns
Load Byte (SD)	50 ns	100 ns
Floating-point Add (HSD)	50 ns	100 ns
Floating-point Multiply (HSD)	50 ns	150 ns
Floating-point Reciprocate (HS)	50 ns	150 ns
Floating-point Square Root (HS)	50 ns	150 ns
Floating-point Reciprocate (D)	350 ns	450 ns
Floating-point Sine (HS)	300 ns	400 ns

Table 1. Selected Mark IIA Scalar Instructions.

Instruction (H-halfword, S-singleword, D-doubleword)	Startup Time	Time per Iteration
for $I := 1$ **to** *SIZE* **do** $X[I] := D$ (H)	250 ns	6.25 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := D$ (S)	250 ns	12.5 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := D$ (D)	250 ns	25 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := X[I] + Y[I]*S$ (H)	250 ns	12.5 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := X[I] + Y[I]*S$ (S)	250 ns	25 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := X[I] + Y[I]*S$ (D)	250 ns	50 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := Y[I] + Z[I]*S$ (H)	250 ns	18.75 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := Y[I] + Z[I]*S$ (S)	250 ns	37.5 ns
for $I := 1$ **to** *SIZE* **do** $X[I] := Y[I] + Z[I]*S$ (D)	250 ns	75 ns

Table 2. Selected Mark IIA Vector Instructions.

Instruction (H-halfword, S-singleword)	Time per Iteration
1024 Point Complex Floating-point FFT (H)	1.3 ms
1024 Point Complex Floating-point FFT (S)	5.2 ms

Table 3. Selected Mark IIA Signal Processing Instructions.

Hardware Description of the Mark IIA

Each S-1 uniprocessor contains a multiple stage "pipe" of instructions-in-process, segmented into 50 ns pipe intervals; that is, each uniprocessor operates at a peak rate of 20 million instructions per second (MIPS) for scalar operations. As Figure 2 indicates, there are two principal modules of an S-1 Mark IIA which implement this pipeline: the IBOX and the ABOX. The function of the IBOX is to prepare and deliver operands and instructions to the ABOX, whose function, in turn, is to perform the requested operation and return the results thereof to the IBOX.

The S-1 Mark IIA IBOX

The IBOX itself consists of four microengines: the F-sequencer, which fetches instructions, the P-sequencer, which decodes instructions, the I-sequencer, which fetches data and prepares operands, and the M-sequencer which serves as a memory interface and control unit. The F-, P-, and I-sequencers each control different portions of the pipeline; the M-sequencer services memory (cache miss) requests from the F- and I-sequencers.

We will not delve deeply in the operation of the IBOX; however, two aspects of its design are relevant to a discussion of signal processing on the Mark IIA. The first is the parallel nature of the data cache in the IBOX; in particular, it can read out eight sequential halfwords of data every 50 ns cycle. This performance level is necessary to achieve the needed vector bandwidth. To allow the arbitrary alignment of operands (scalars and vectors) there is an alignment network; this is combined with a small buffer memory to implement a "operand queue" to both align and buffer the deliver of data from the IBOX to the ABOX. This operand queue has a special function for signal processing. We have developed an algorithm for parallel "bit reversal" of vectors using this memory [5] which allows the "bit reversal" operation to

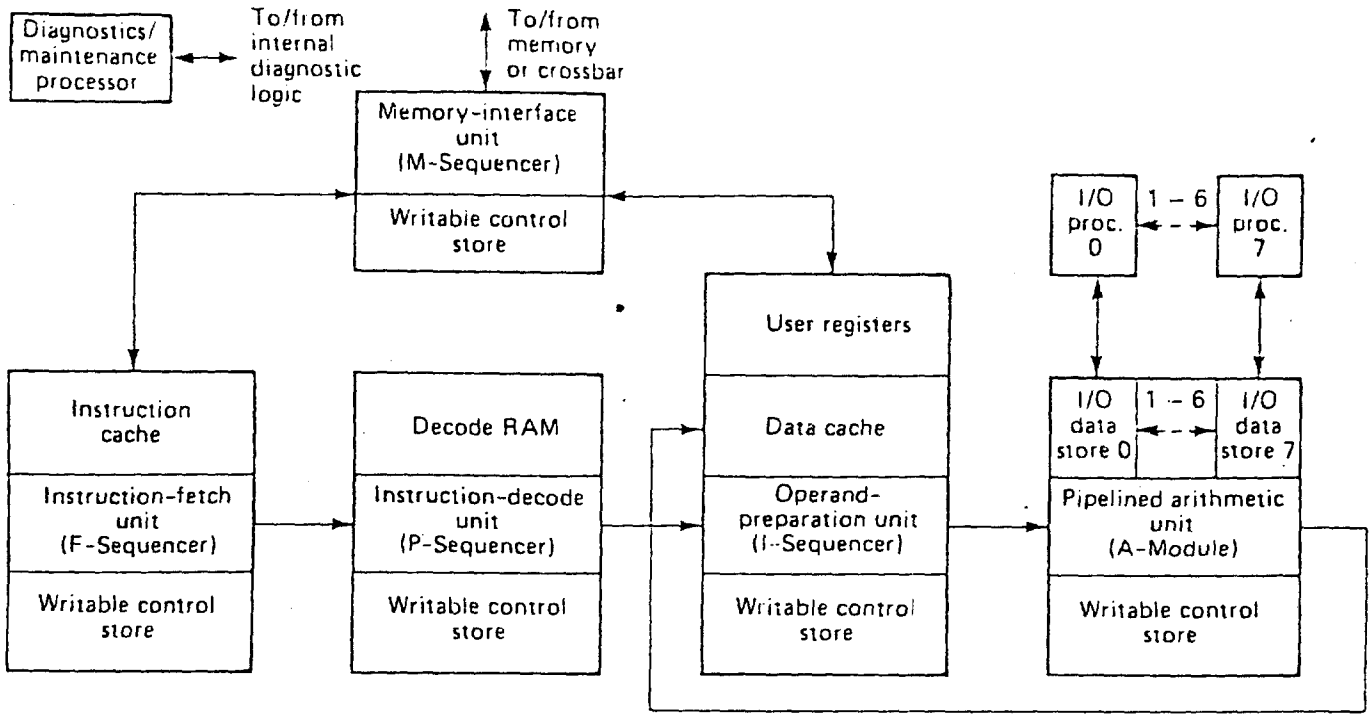

Figure 2. S-1 Mark IIA Uniprocessor Functional Diagram.

proceed at the same rate as a vector move.

The S-1 Mark IIA ABOX

We will limit our discussion of the ABOX to its arithmetic capabilites; however, a few explanations are in order. The ABOX internally works with a uniform bus format (the "internal" format); thus we occasionally refer to the S-1 architecture format as the external format. The main relevence of this to signal processing is in the increased accuracy with which many functions can be computed. This extra accuracy is necessary for the "good to the last bit" evaluation of many of the ABOX special functions (an example would be the SINE instruction). In addition, the evaluation of "composed" functions (i.e. a function formed by the composition of more than one basic operation) can be evaluated with less rounding error; this is important in the execution of a FFT or a dot product. Note that all cycle time references in the ABOX description are to 25 ns cycles, since the ABOX runs at twice the rate of the IBOX.

The arithmetic hardware of the ABOX consists of a Multiplier Functional Unit and an Adder Functional Unit. The term "functional unit" refers to its independent nature, i.e. the adder and multiplier are capable of simultaneous operation. The multiplier has six pipeline stages (for a total latency of 150 ns) and the adder has four (for a latency of 100 ns); both can produce a result every (25 ns) cycle.

Multiplier Functional Unit The multipler functional unit includes four independent 18x36 bit multiplier arrays and two half-word integer or floating-point adders, all of which are capable of simultaneous operation.

These are interconnected in various ways for executing different instructions. Some examples of such variability are two cycles of all 4 arrays to do a double-word integer or floating-point multiplication, one cycle of two arrays for a single-word integer or floating-point multiplication and one cycle of all four arrays plus two half-word adders to do a complex half-word multiplication.

Division is accomplished within the multiplier by one of two methods, both involving reciprocation. To compute single-word y/x, we first calculate $1/x$ by a piecewise quadratric approximation initiated with a table-lookup value [6]. With the tables included in the Mark IIA, this method produces about 30 bits

of precision for $1/x$. We may then rapidly calculate a slightly imprecise value of y/x by multiplication; a precise result requires an additional multiplication, in order that correct rounding may be guaranteed. For double-word division, we compute the reciprocal as before to 30 bits and then use a single Newton iteration to double the precision. As before, an additional multiplication is necessary for precise division. (For integer double-word division, in which 72-bits are involved, we must iterate twice).

The piecewise quadratic approximation techniques employed to calculate for reciprocals also work well for many other functions [6]. The S-1 Mark IIA ABOX implements sine, cosine, arctangent, exponential (2^x, e^x, and y^x), logarithm (lg(x), ln(x), and log(x)), and erf (error function) using this method; thus, the calculation of these functions can be pipelined at 50 ns per result.

Adder Functional Unit The adder functional unit is capable of doing floating-point or integer adds of all precisions, shifts, byte operations, boolean, bit counting, etc. It consists of four pipe stages and can produce a new result every cycle. The floating-point addition algorithm it employs is fairly novel [7], but discussion of it is deferred to the reference for reasons of brevity and its secondary relevance to signal processing per se.

S-1 Multiprocessor

There are many signal processing tasks which are too compute-intensive to be effectively serviced by current computing systems. (Examples include real-time speech processing, radar and sonar imaging and discrimination.) The S-1 multiprocessor is an attempt to make some of these tasks more feasible in the near term. By combining extremely high-performance uniprocessors (the Mark IIA and future S-1 generations) into a medium population multiprocessor (e.g. 16 processors), many compute-intensive algorithms can be run from 10 to 100 times faster than on the best currently available systems.

The most straightforward use of a multiprocessor system is to dedicate its separate processors to performance of different tasks ("task-pipelining"). An obvious example of such a division would be data collection, preprocessing, analysis, and postprocessing in a typical real time signal processing enviornment. When this level of task partitioning fails to achieve the desired performance speed-up, one may use multiple processors in confederation to perform single tasks.

For instrance, one could just go buy a hundred low-cost signal processors, attach them to a host computer and claim a 100-fold increase in performance over a single system. Unfortunately, it has been found (more than once) that the "little" things become important when striving for very high degrees of parallelism. The inherently serial parts of the algorithm of interest will dominate as the parallel loops are made to run ever faster, and when new algorithms which reduce the serial sections are implemented, it is usually found that the interprocessor communication begins to dominate execution time. This is because even though the cost of one signal processor talking to one host is (relatively) insignificant, the cost of 100 doing so typically is not.

The S-1 architecture and the Mark IIA implementation have been designed with multiprocessing in mind. The logical structure of the S-1 multiprocessor is depicted in Figure 3. The member processors share a large (16 billion words) common physical address space with all accesses to the shared memory being mediated by the Crossbar Switch. The use of caches is important in reducing contention for memory access by two or more processors. With a 99% "hit rate," the Switch data traffic of a multiprocessor is reduced by a factor of 100. This extremely high hit rate is not at all unlikely: we have seen it in detailed simulations, it compares favorably with the measured hit rates of the S-1 Mark I (95-98% using a 4K cache), and it agrees well with the predictions of [8].

With the use of shared memory and special interprocessor communication instructions, relatively tight coupling between member processors of medium-sized multiprocessors can readily be achieved. When using an S-1 multiprocessor system in a "task-pipelined" manner, results from previous stages may be passed directly to the next stage without first being evicted from main memory, without transmission over bandwidth-limited buses, and without delaying the execution of the current stage of processing on the next batch of data. We have also been working on the more complicated effort of using an S-1 multiprocessor for efficient execution of a single task; preliminary studies indicate that applications considered tightly-coupled may nonetheless be able to achieve a speed-up factor of more than ten on a sixteen-processor system ([9]).

Summary

The S-1 Mark IIA uniprocessor (and future implementations of the S-1 architecture) will provide

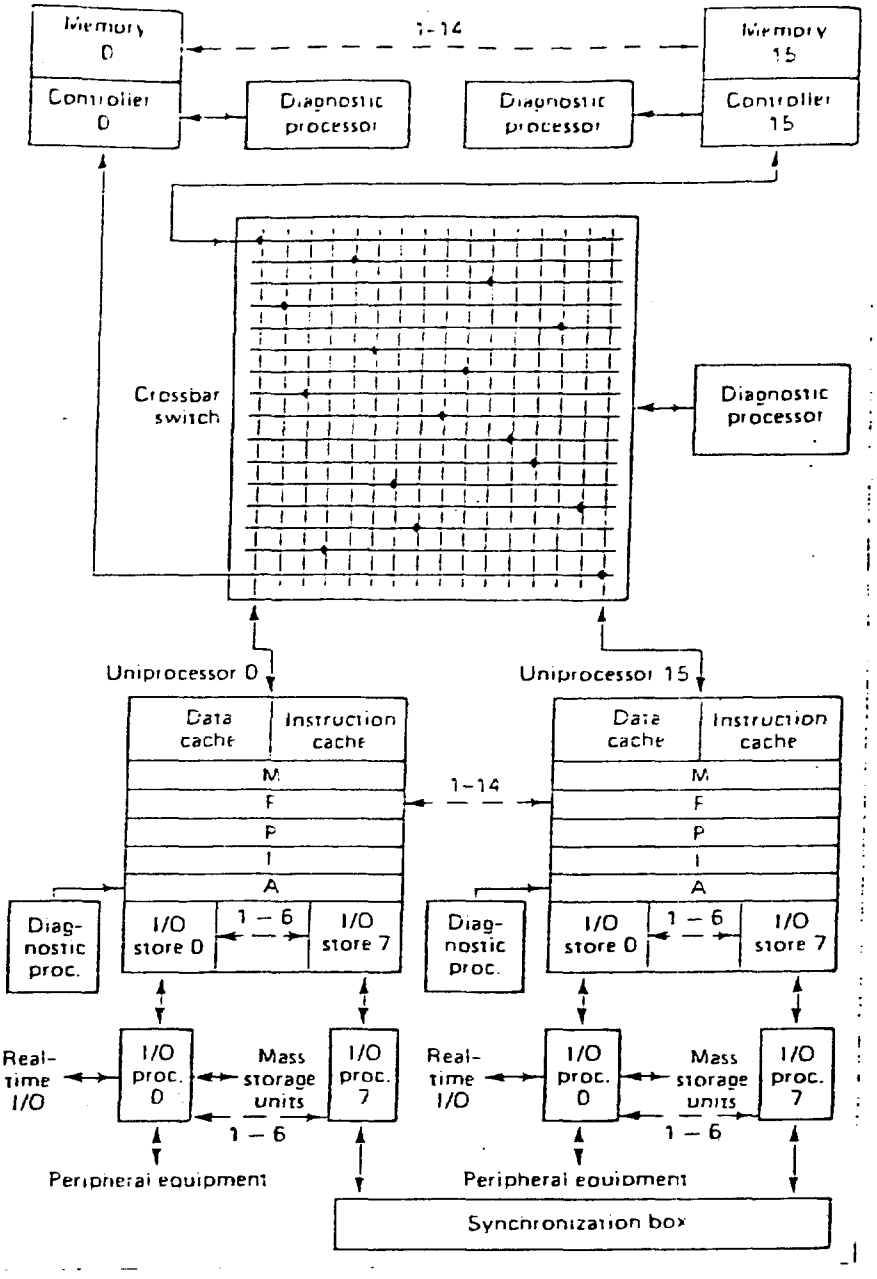

Figure 3. S-1 Multiprocessor configuration.

a level of compute power, programmability and cost-effectiveness which will allow the performance of signal processing tasks not feasible with current systems. The combination of general-purpose and signal processing architectures in one machine is expected to be particularly valuable in reducing signal processing system costs and creation latencies. For those tasks which are too demanding even for such a powerful processor, the S-1 Mark IIA multiprocessor system will improve the total available compute power by more than an order-of-magnitude, while retaining the unusually high programmability and cost-effectiveness of the uniprocessor.

Acknowledgments

We wish to express our appreciation to our colleagues in the S-1 effort for their many valuable contributions to the research reported here.

The S-1 Project is sponsored by the Naval Material Command. This work was performed under the supervision of the Office of Naval Research and the Naval Electronic Systems Command. The Lawrence Livermore National Laboratory is operated by the University of California for the U. S. Department of Energy under Contract W-7405-eng-48.

References

1. L.C. Widdoes, "The S-1 Project: Developing High-Performance Digital Computers",*IEEE Computer Society: COMPCOM Spring 1980.*

2. Staff, *S-1 Project FY1979 Annual Report*, University of California, Lawrence Livermore National Lab., UCID 18619 (1979)

3. J.T. Coonen, *Specifications for a Proposed Standard for Floating Point Arithmetic*, University of California at Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M78/72 (1978)

4. C.J. Weinstein and A.V. Oppenheim, "A Comparison of Roundoff Noise in Floating Point and Fixed Point Digital Filter Realisation", *Proc. IEEE, June, 1969*

5. P. Michael Farmwald, "Parallel Transposition and Bit Reversing using Interleaved Memory", to be submitted to *IEEE Trans. on Computers*

6. P. Michael Farmwald, "High Bandwidth Evaluation of Elementary Functions", to be submitted to *IEEE Trans. on Computers*

7. P. Michael Farmwald, "Faster Binary Floating-Point Adders", to be submitted to *IEEE Trans. on Computers*

8. K. R. Kaplan and R. O. Winder, "Cache-Based Computer Systems", *Computer*, Vol. 6, No. 3, March 1973

9. Erik J. Gilbert, *An Investigation of the Partitioning of Algorithms Across an MIMD Computing System*, Lawrence Livermore National Laboratory Report, Febuary 1980